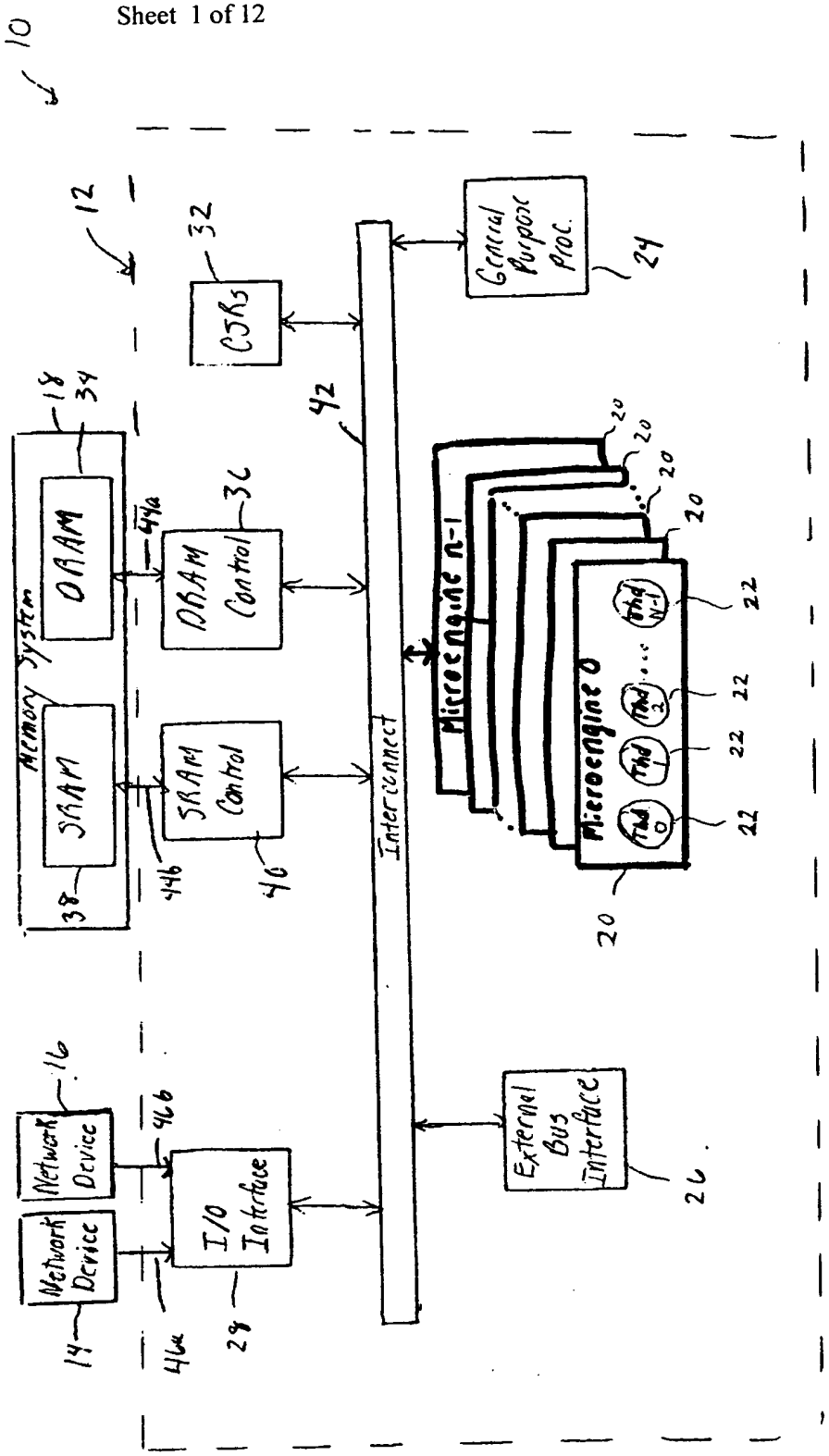
10
12
Memory System
18
34
DRAM
SRAM
38
46a
46b
46c
46d
46e
DRAM Control
SRAM Control
40
42
Interconnect
CSRs
32
36
General Purpose Proc.
24
Microengine n-1
Microengine 0
Thd 0
Thd 1
Thd 2
Thd n-1
20
22
External Bus Interface
26
I/O Interface
28
Network Device
16
14

FIG. 1

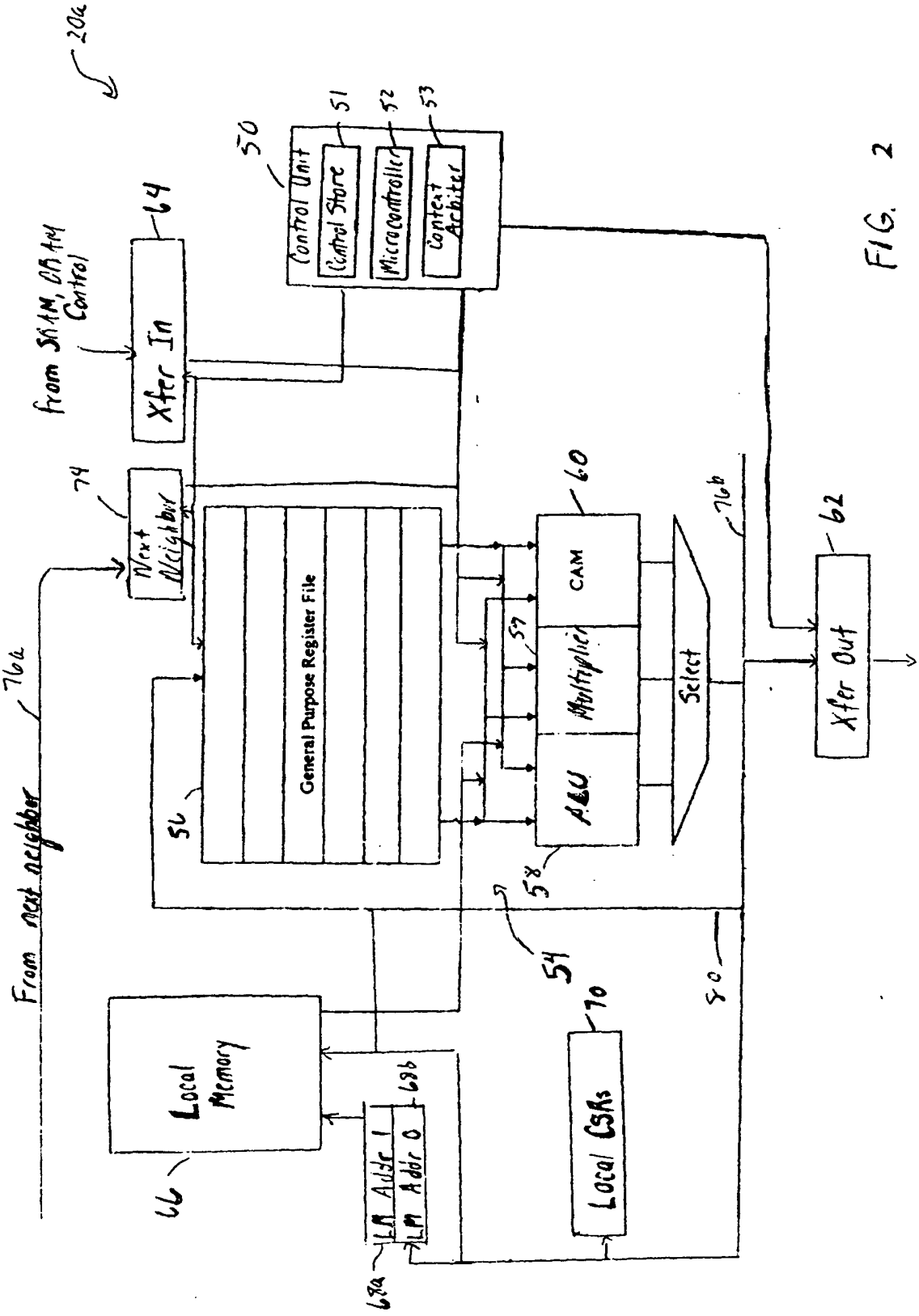
20a
50
Control Unit
51
Control Store
52
Microcontroller
53
Context Arbiter
64
Xfer In
From SRAM, DRAM Control
74
Next Neighbor
From next neighbor
76a
56
General Purpose Register File
60
CAM
Multiplier
ALU
58
57
76b
Select
62
Xfer Out
To SRAM, DRAM Control
54
52
66
Local Memory
68b
LM Addr 1
LM Addr 0
68a
70
Local CSRs

FIG. 2

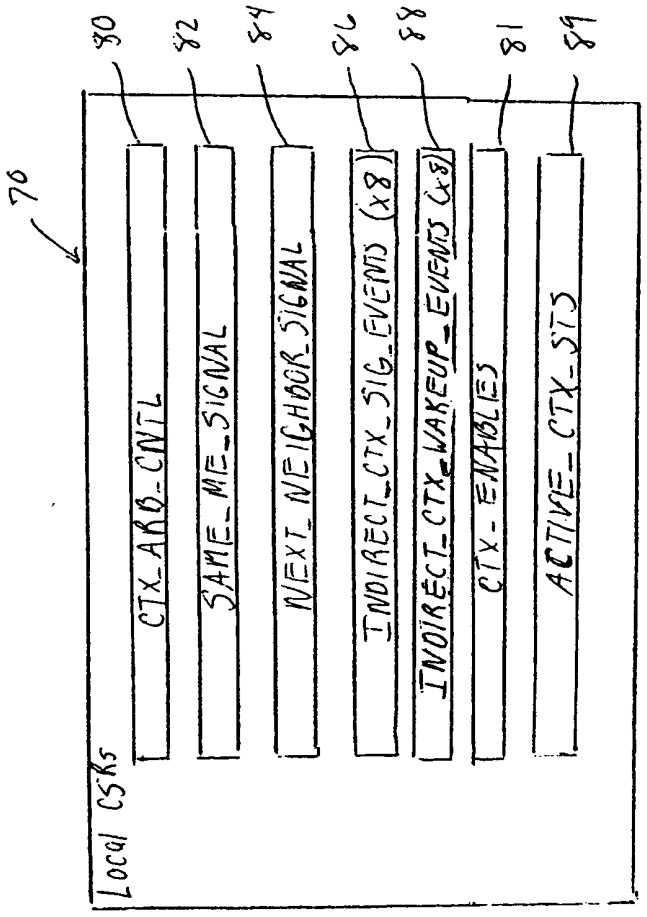
70
80
82
84
86
88
81
89
CTX_ARB_CNTL
SAME_ME_SIGNAL
NEXT_NEIGHBOR_SIGNAL
INDIRECT_CTX_SIG_EVENTS (x8)
INDIRECT_CTX_WAKEUP_EVENTS (x8)
CTX_ENABLES
ACTIVE_CTX_STS
Local CSRs

FIG. 3

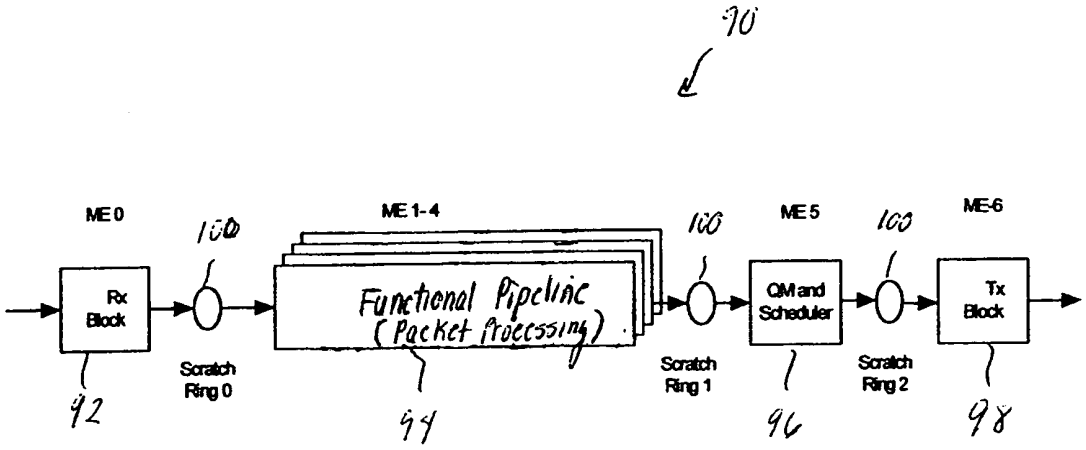
90
ME 0
100
ME 1-4
100
ME 5
100
ME-6
Rx Block
Functional Pipelinc (Packet Processing)
QM and Scheduler
Tx Block
Scratch Ring 0
Scratch Ring 1
Scratch Ring 2
92
94
96
98

FIG. 4

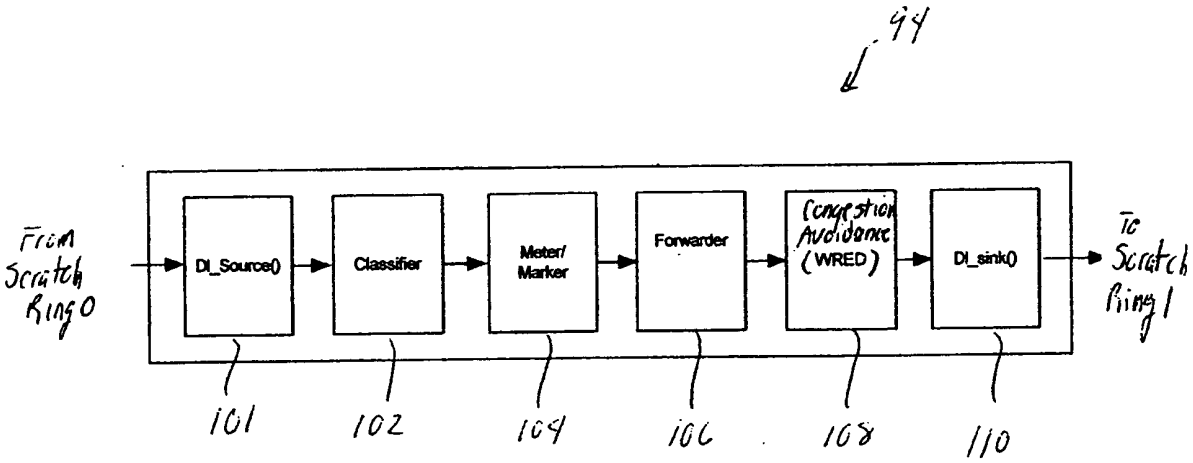
94
From Scratch Ring 0
DI_Source()
Classifier
Meter/ Marker
Forwarder
Congestion Avoidance (WRED)
DI_sink()
To Scratch Ring 1
101
102
104
106
108
110

FIG. 5

120

Register Name	Latency in terms of instructions			Usage Latency Comments
	Write	Read	Usage	
SAME_ME_SIGNAL	3	2	8	The same ME will be signaled 8 cycles after the CSR write.

122

FIG. 6

130

```
wred ()
{
        if (ctx () == 0)
        {
        //Wait for signal from previous ME and thread 7
        wait_for_all (&next_thread_signal, &wred_next_me_sig);
                cam_clear ();
        }
        else
        {
                wait_for_all (&next_thread_signal);
        }
        .....
        .....
        // WRED packet processing
        .....
        signal_next_thread ()  // Instruction 1
        .....
        // There is a minimum of 3 cycles delay between instruction 1 and instruction 2
        // to allow the signal to propagate and to ensure thread execution sequence.
        .....
        // Wait for previous thread signal
        wait_for_sig (&sig);    // Instruction 2
        .....
        if (ctx () == 7)
        {
                // Signal next ME
                cap_fast_write (wred_me_sig_csr, csr_interthread_sig);
        }
        else
        {
                // The thread gives up the context voluntarily at this point to ensure that
                // thread 7 gets control as early as possible.  If no context swap occurs
                // here the thread would continue to execute non-critical section code or
                //          next microblock, thereby delaying thread 7 getting the control.
                ctx_arb (voluntary);
        }

        // Critical section processing ends

        // Non-critical section code or code for next microblock begins
}
```

132

134

136

138

140

FIG. 7

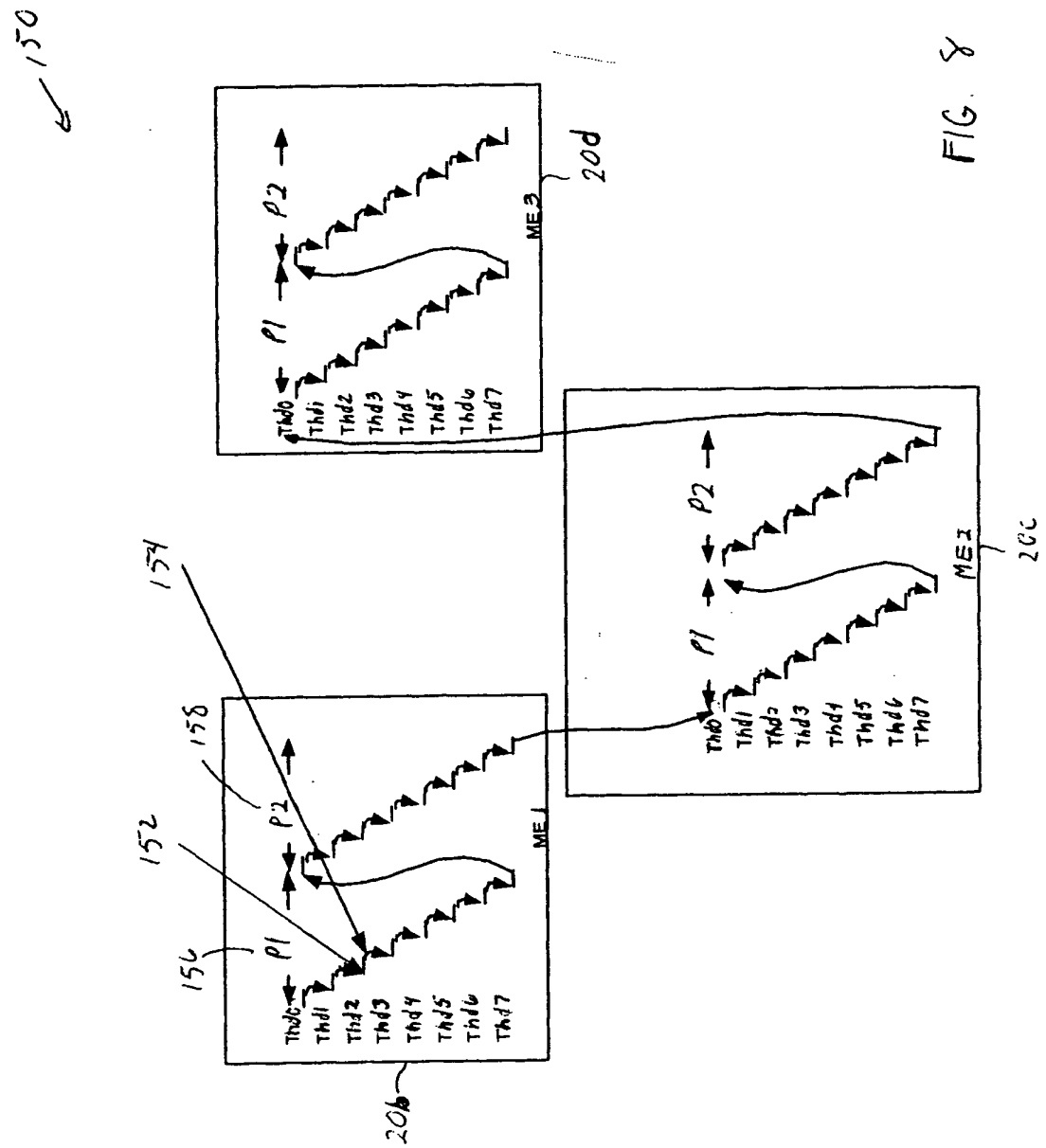
150
152
154
156
158
P1
P2
Thd0
Thd1
Thd2
Thd3
Thd4
Thd5
Thd6
Thd7
ME1
ME2
ME3
20b
20c
20d

FIG. 8

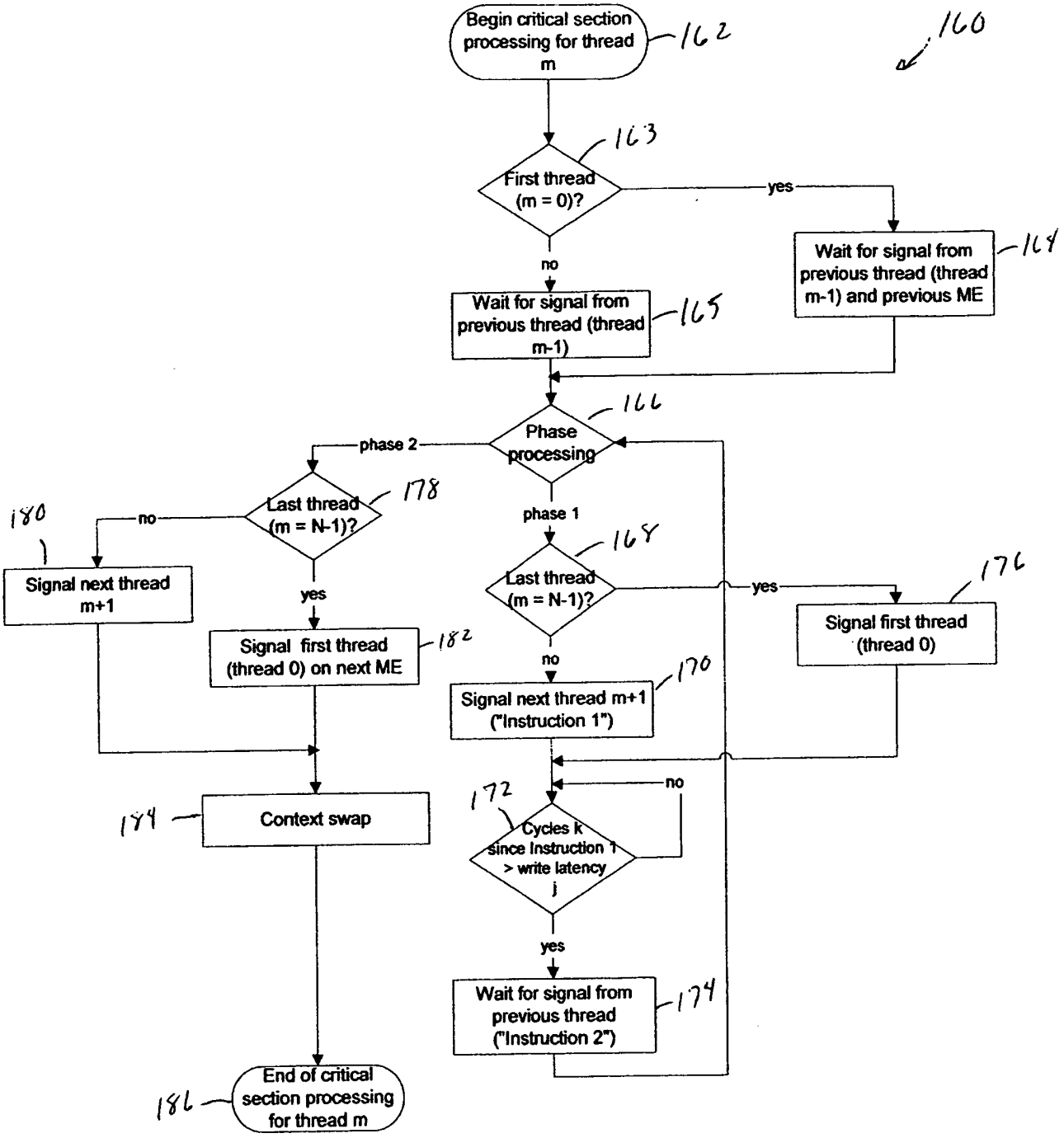
Begin critical section processing for thread m
162
160
First thread (m = 0)?
163
yes
Wait for signal from previous thread (thread m-1) and previous ME
164
no
Wait for signal from previous thread (thread m-1)
165
Phase processing
166
phase 2
phase 1
Last thread (m = N-1)?
178
180
no
yes
Signal next thread m+1
Signal first thread (thread 0) on next ME
182
Last thread (m = N-1)?
168
yes
176
Signal first thread (thread 0)
no
Signal next thread m+1 ("Instruction 1")
170
Context swap
184
172
Cycles k since Instruction 1 > write latency j
no
yes
Wait for signal from previous thread ("Instruction 2")
174
End of critical section processing for thread m
186

FIG. 9

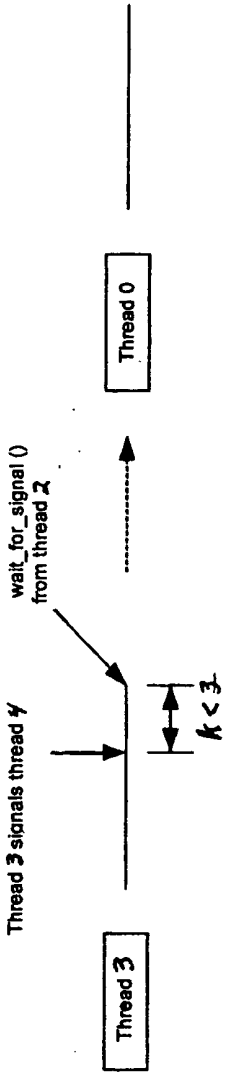

FIG. 10A

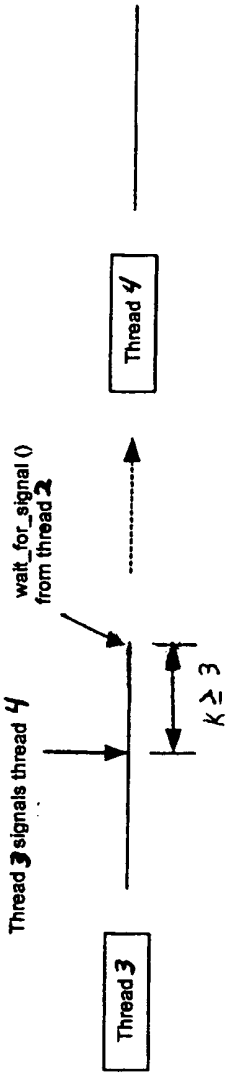

FIG. 10B

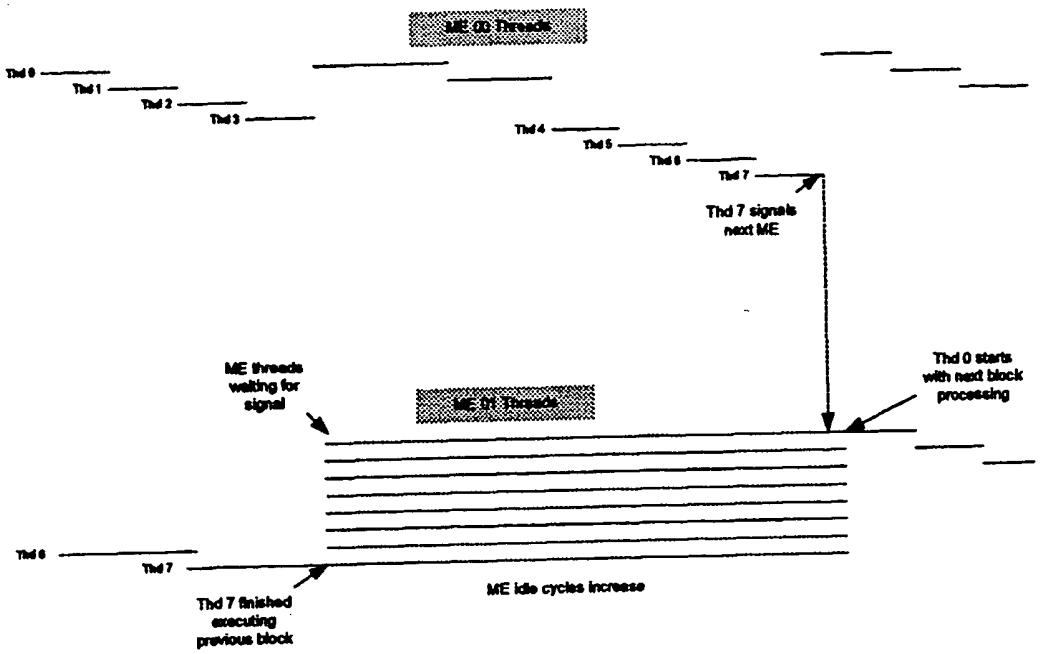

FIG. 11A

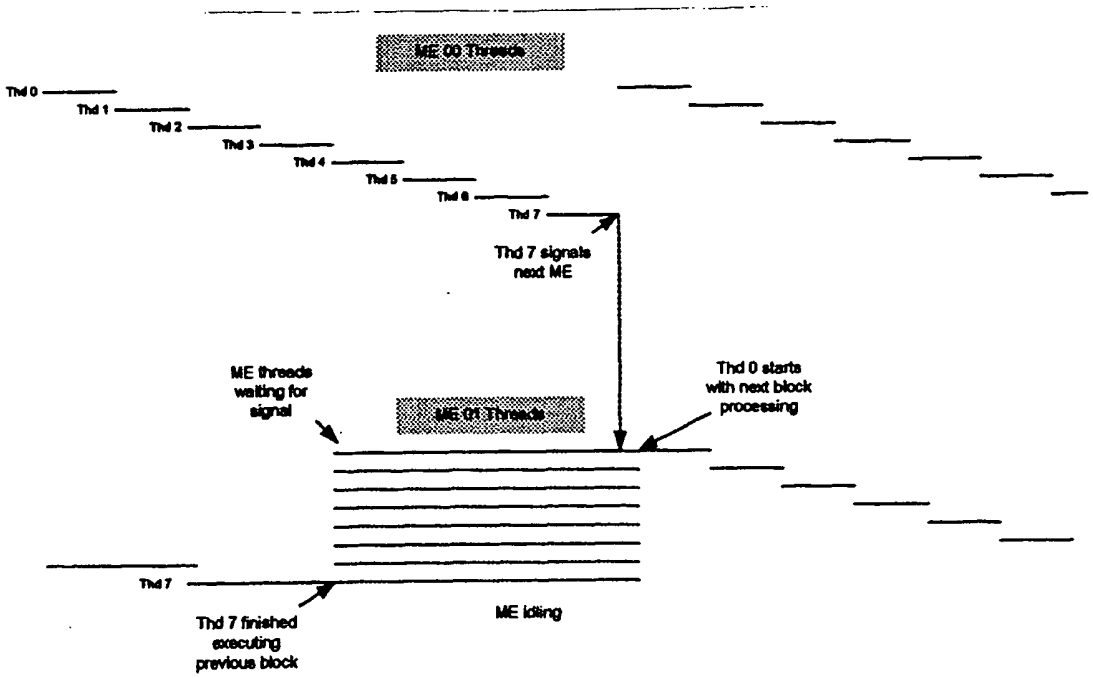

FIG. 11B

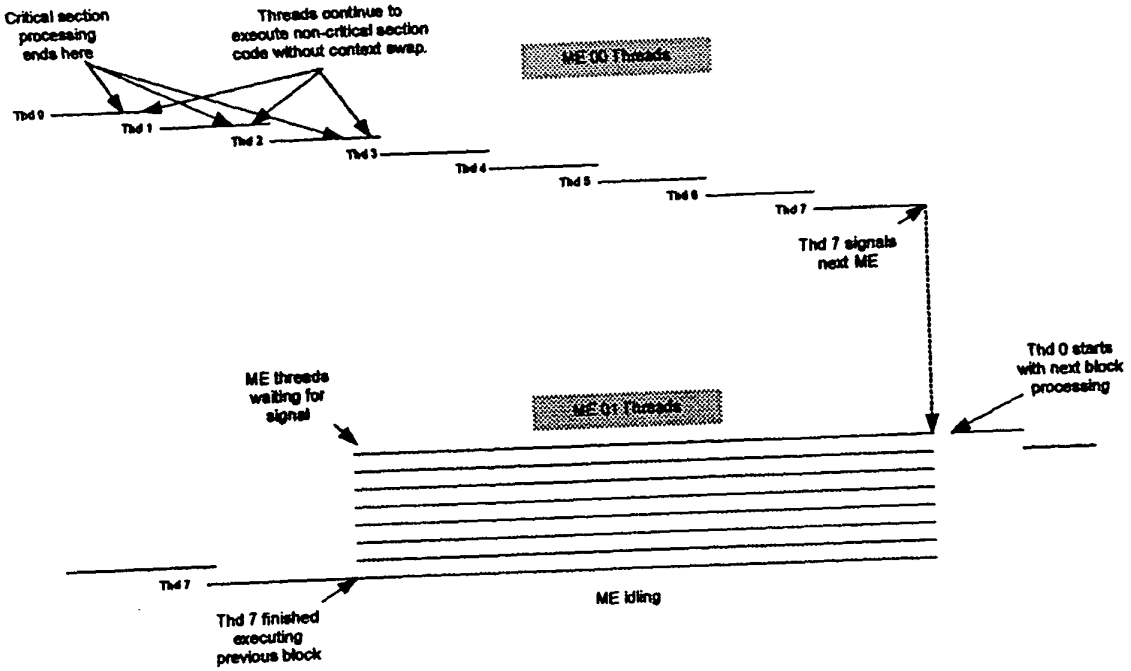

FIG. 12A

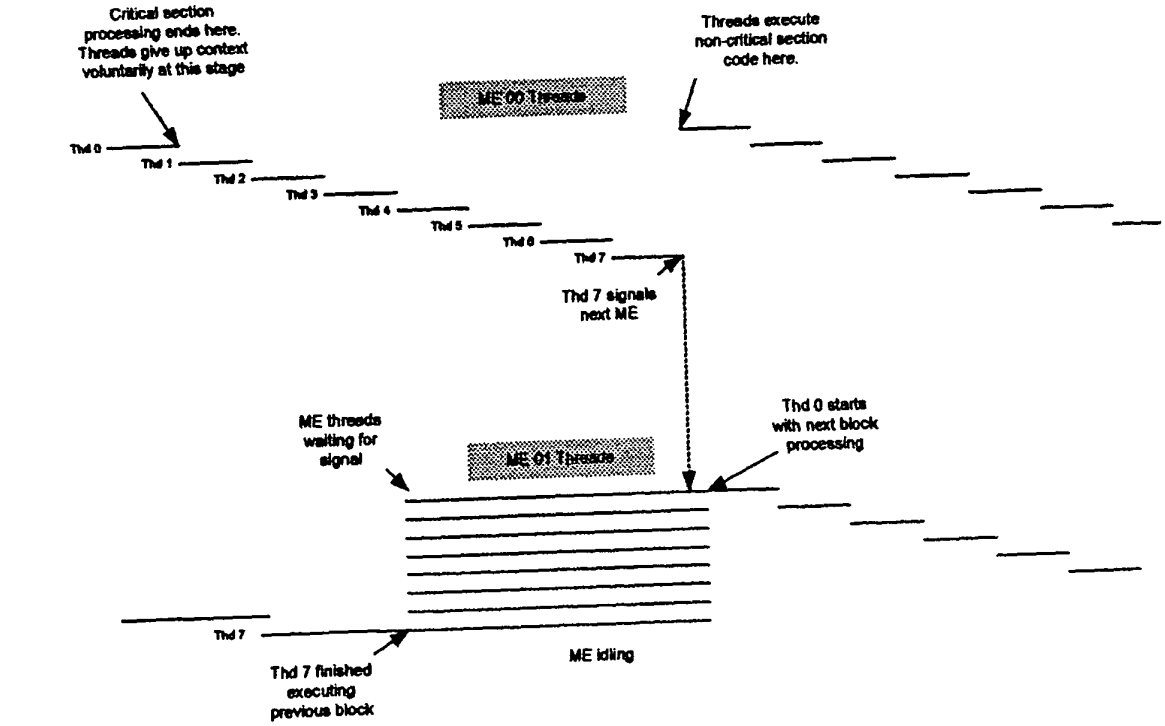

FIG. 12B

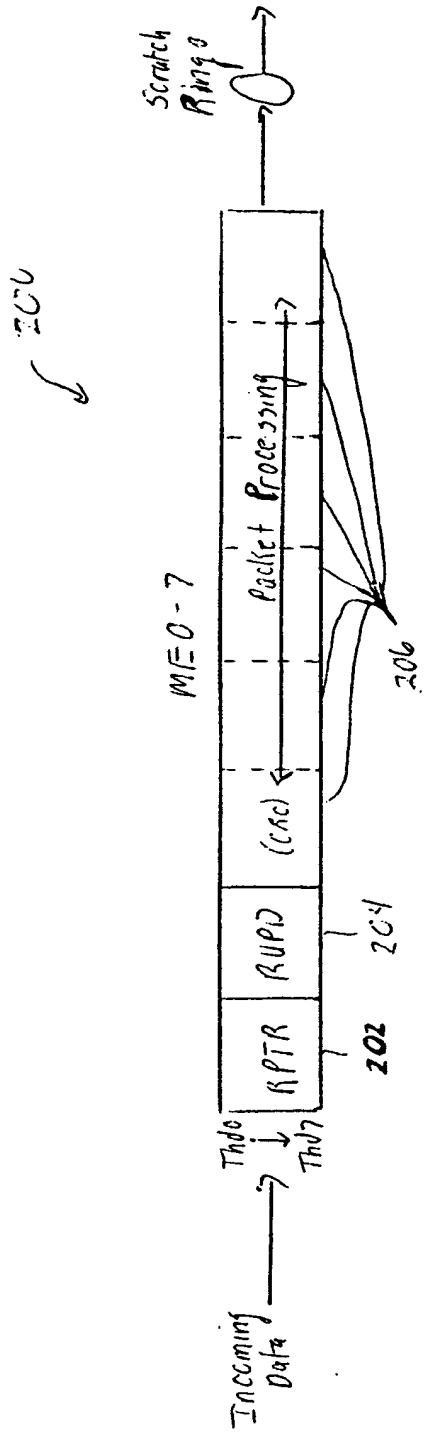
Scratch Ring 0
206
ME 0 - 7
Packet Processing
(CRC)
RUPD
RPTR
202
204
Thd0
Thd7
Incoming Data

FIG. 13